Homework ada4

TMI M1 37-176839 Koichiro Tamura

homework1

線形モデル

$$f_\theta(\mathbf{x}) = \sum_{j=1}^{b} \theta_j \phi_j(\mathbf{x})$$

に対する重み付き最小二乗法

$$\min_\theta \frac{1}{2} \sum_{i=1}^{n} \tilde{w}_i (f_\theta(\mathbf{x_i}) - y_i)^2$$

の解が次式で与えられることを示せ.

$$\hat{\theta} = (\mathbf{\Phi}^T \tilde{\mathbf{W}} \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \tilde{\mathbf{W}} \mathbf{y}$$

proof:

$$\begin{aligned} L &= \frac{1}{2} \sum_{i=1}^{n} \tilde{w}_i (f_\theta(\mathbf{x_i}) - y_i)^2 \\ &= \frac{1}{2} \tilde{\mathbf{W}} (\mathbf{\Phi}\theta - \mathbf{y})^T (\mathbf{\Phi}\theta - \mathbf{y}) \\ \therefore \nabla_\theta L &= \mathbf{\Phi}^T \tilde{\mathbf{W}} \mathbf{\Phi} \theta - \mathbf{\Phi}^T \tilde{\mathbf{W}} \mathbf{y} = 0 \\ \therefore \hat{\theta} &= (\mathbf{\Phi}^T \tilde{\mathbf{W}} \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \tilde{\mathbf{W}} \mathbf{y} \end{aligned}$$

【Q.E.D】

homework2

微分可能で対象な損失$\rho(r)$に対して$\tilde{r}$で接する二次上界は存在するなら次式で与えられることを示せ.

$$\tilde{\rho} = \frac{\tilde{w}}{2} r^2 + const$$

$$\tilde{w} = \frac{\rho'(\tilde{r})}{\tilde{r}}$$

proof:

対象な2次上限は

$$\tilde{\rho}(r) = ar^2 + b$$

と与えられる.

このとき，$\tilde{r}$で接するので,

$$\rho'(\tilde{r}) = f'(\tilde{r})$$

$$\therefore \rho'(\tilde{r}) = 2a\tilde{r}$$

$$\therefore a = \frac{\rho'(\tilde{r})}{2\tilde{r}}$$

$$\therefore \tilde{\rho}(r) = \frac{\tilde{w}}{2} r^2 + const$$
$$\tilde{w} = \frac{\rho'(\tilde{r})}{\tilde{r}}$$

【Q.E.D】

homework3

直線モデル$f_\theta(x) = \theta_1 + \theta_2 x$に対して，テューキー回帰の繰り返し最小二乗アルゴリズムを実装せよ

In [1]:

```
%matplotlib inline

import numpy as np
import math
import random
import matplotlib.pyplot as plt
```

In [2]:

```
def func(x):
    return x
```

In [3]:

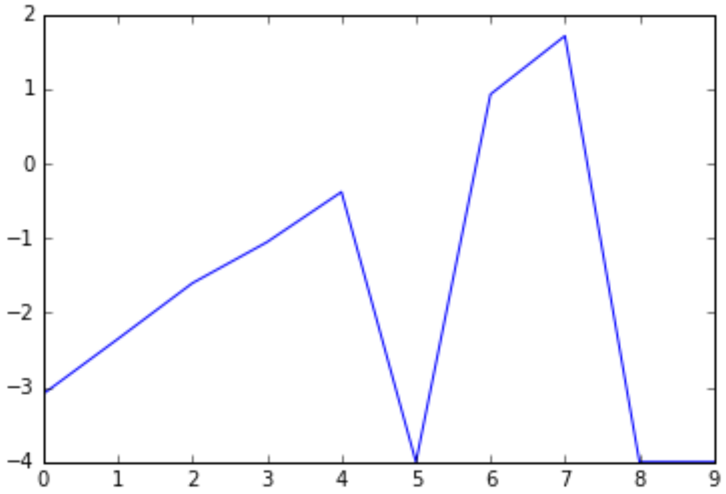
```
# データセット
train_x = np.linspace(-3, 3, 10)
# np.random.shuffle(train_x)
train_y = np.array([func(train_x[i]) for i in range(len(train_x))] + (np.random.rand(len(train_x)) - 0.5)*(
train_y[5] = -4
train_y[8] = -4
train_y[9]  = -4
```

In [4]:

```
plt.plot(train_y)
```

Out[4]:

[<matplotlib.lines.Line2D at 0x10cb7d0f0>]



In [5]:

```
# cal phi
Phi =  np.ones(2*len(train_x)).reshape([len(train_x), 2])

for i in range(len(train_x)):
        Phi[i, 1] = train_x[i]
# Phi
```

In [6]:

```
# initialize W
W = np.zeros([len(train_x), len(train_x)])
for i in range(len(train_x)):
    W[i, i] = 1/6
# W
```

In [7]:

```
def get_theta(Phi, W, train_y):
    return np.linalg.inv(Phi.T.dot(W).dot(Phi)).dot(Phi.T).dot(W).dot(train_y)
```

In [8]:

```
theta = get_theta(Phi, W, train_y)
```

In [9]:

```
theta
```

Out[9]:

array([-1.78370696, -0.00804929])

In [10]:

```
def predict(train_x, theta):
    return 1* theta[0] + train_x*theta[1]
```

In [11]:

```
def update_W(W, theta, train_x, train_y, eta=1):
    r = predict(train_x, theta) - train_y
    for i in range(len(r)):
        if np.abs(r[i]) > eta:
            W[i][i] = 0
        else:
            W[i][i] = ((1-r[i]**2)/eta**2)**2
    return W
```

In [12]:

```
for step in range(100):
    W = update_W(W, theta, train_x, train_y)
    # print(W.shape)
    theta = get_theta(Phi, W, train_y)
```

In [13]:

```
predictions = predict(train_x, theta)
plt.plot(predictions)
```

Out[13]:

[<matplotlib.lines.Line2D at 0x10d0b3e48>]